Section overview

Markov Decision Process

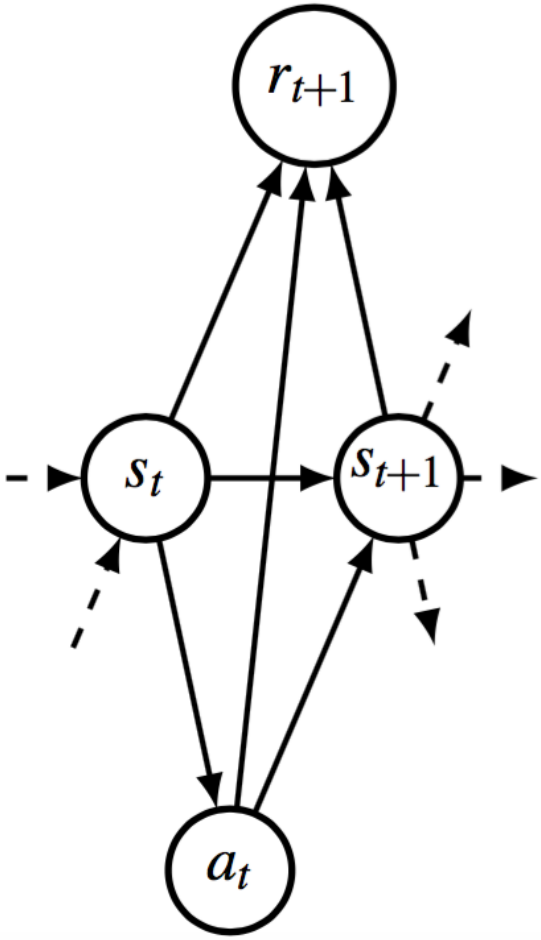
Markov Decision Process (MDP)

Definition (Markov Decision Process)

- $\mathcal{S}$: State space
- $\mathcal{A}$: Action space
- $\mathcal{P}^a_{ss'}$: Transition probability $p(s_{t+1}|s_t, a_t)$
- $\gamma \in [0, 1]$: Discount factor
- $\mathcal{R}^a_{ss'} = r(s, a, s')$ immediate/instantaneous reward function.
  In temporal notation[a] $r_{t+1} = r(s_{t+1}, s_t, a_t)$
- π: Policy
  - Stochastic: $\boldsymbol{a} \sim p_\pi(\boldsymbol{a}|\boldsymbol{s})$
    alternative notations:
    $p_\pi(\boldsymbol{a}|\boldsymbol{s}) = \pi(\boldsymbol{a}|\boldsymbol{s}) \equiv \pi(a, s)$ (probability)
  - Deterministic: $\boldsymbol{a} = \pi(\boldsymbol{s})$ (indicator function)

[a] This means the reward is collected upon the transition from s_t to s_{t+1}, which we determine to occur at time point $t + 1$.

Graphical model of an MDP



Circles denote variables, edges denote conditional dependences between variables.

Value function for MDPs

How good is it to be in a given state?

Definition (Value function)

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[R_t | S_t = s\right] = E[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s] \tag{21}$$

where R_t is a discounted total return and the r_{t+k+1} are immediate rewards.

We will see how the value function is a self-consistent linear equation, in analogy to what we derived for the MRP.

Value function self-consistency

$$V^{\pi}(s) \quad = \quad \mathbb{E}_{\pi}\left[R_t \middle| S_t{=}s\right]$$

Value function self-consistency

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}\left[R_t | S_t{=}s\right] \\
&= \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t{=}s\right] \\
&= \mathbb{E}_{\pi}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | S_t{=}s\right] \\
&= \mathbb{E}\left[r_{t+1} | S_t{=}s\right] + \gamma \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | S_t{=}s\right]
\end{aligned}
$$

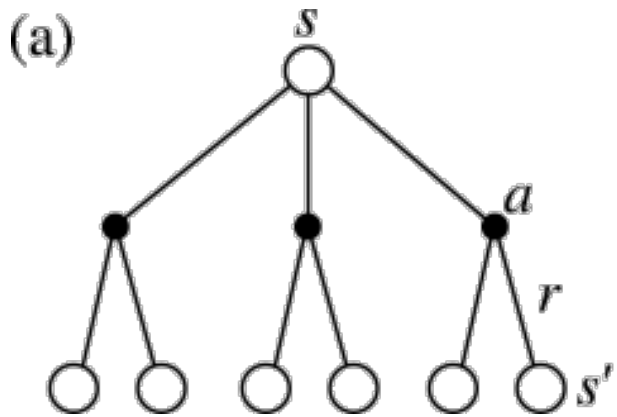
Value function self-consistency

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}\left[R_t | S_t{=}s\right] \\
&= \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t{=}s\right] \\
&= \mathbb{E}_{\pi}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | S_t{=}s\right] \\
&= \mathbb{E}\left[r_{t+1} | S_t{=}s\right] + \gamma \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | S_t{=}s\right]
\end{aligned}
$$

To understand the next step better, let us unpack the meaning of the two expectation terms $\mathbb{E}\left[\ldots\right]$.

Backup diagrams

Figure (a) shows a backup diagram: We use the diagram to trace the paths from state s to successor states s' given that we chose an action a. We use these to transfer state value information **back** to a state s from its successor states s'. This is the update or backup operation that forms the heart of reinforcement learning methods.



Backups and value function unpacking

We can use the intuition from this diagram, to "**backup**" the current state value from successor states. To be able to calculate $v(s)$ we only need to average over all possible traces and their rewards.

Backups and value function unpacking

We can use the intuition from this diagram, to "**backup**" the current state value from successor states. To be able to calculate $v(s)$ we only need to average over all possible traces and their rewards.

- the chosen action a (given by policy $P[a|s] = \pi(a, s)$)

Backups and value function unpacking

We can use the intuition from this diagram, to "**backup**" the current state value from successor states. To be able to calculate $v(s)$ we only need to average over all possible traces and their rewards.

- the chosen action a (given by policy $P[a|s] = \pi(a, s)$)
- the probability of transition to s' (give by transition probability $P[s'|s, a] = \mathcal{P}^a_{ss'}$)

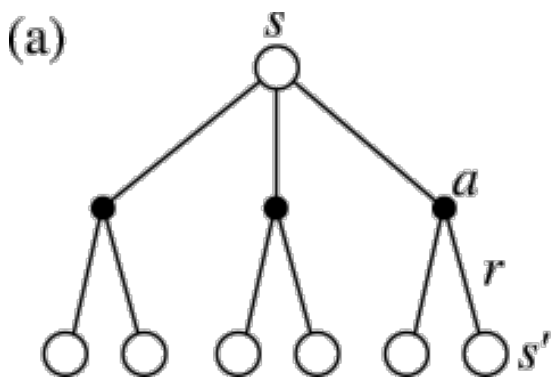
Backups and value function unpacking

We can use the intuition from this diagram, to "**backup**" the current state value from successor states. To be able to calculate $v(s)$ we only need to average over all possible traces and their rewards.

- the chosen action a (given by policy $P[a|s] = \pi(a, s)$)
- the probability of transition to s' (give by transition probability $P[s'|s, a] = \mathcal{P}^a_{ss'}$)
- the instantaneous reward r (given by reward function $r(s, a, s') = \mathcal{R}^a_{ss'}$)

Backups and value function unpacking

We can use the intuition from this diagram, to "**backup**" the current state value from successor states. To be able to calculate $v(s)$ we only need to average over all possible traces and their rewards.



- the chosen action a (given by policy $P[a|s] = \pi(a, s)$)
- the probability of transition to s' (give by transition probability $P[s'|s, a] = \mathcal{P}^a_{ss'}$)
- the instantaneous reward r (given by reward function $r(s, a, s') = \mathcal{R}^a_{ss'}$)
- the value of state s' (hint this is actually $v(s')$, a recursive definition)

We can now derive the probabilities and values of the paths in the diagram.

Backups and value function unpacking

$$\mathrm{E}[r_{t+1}|S_t = s] = \sum_{a\in\mathcal{A}} P[a|s] \left(\sum_{s'\in\mathcal{S}} P[s'|s,a]\, r(s,a,s')\right)$$
$$\mathrm{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2}|S_t{=}s\right] = \sum_{a\in\mathcal{A}} P[a|s] \left(\sum_{s'\in\mathcal{S}} P[s'|s,a] \sum_{k=0}^{\infty} \gamma^k r_{t+k+2}\right)$$

We also have (from the definition of the value function):

$$\begin{aligned} V^{\pi}(s') &= \mathbb{E}\left[R_{t+1}|S_{t+1} = s'\right] \\ &= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2}|S_{t+1} = s'\right] \end{aligned}$$

With this information we can now to proceed unpack the value function.

Value function self-consistency

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[R_t | S_t = s\right]$$

Value function self-consistency

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}\left[R_t | S_t = s\right] \\
&= \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s\right] \\
&= \mathbb{E}_{\pi}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | S_t = s\right] \\
&= \sum_{a \in \mathcal{A}} \pi(a, s) \left(\sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | S_{t+1} = s'\right]\right)\right) \\
&= \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')\right)
\end{aligned}
$$

Value function self-consistency

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}\left[R_t | S_t = s\right] \\
&= \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s\right] \\
&= \mathbb{E}_{\pi}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | S_t = s\right] \\
&= \sum_{a \in \mathcal{A}} \pi(a, s) \left(\sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | S_{t+1} = s'\right]\right)\right) \\
&= \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')\right)
\end{aligned}
$$

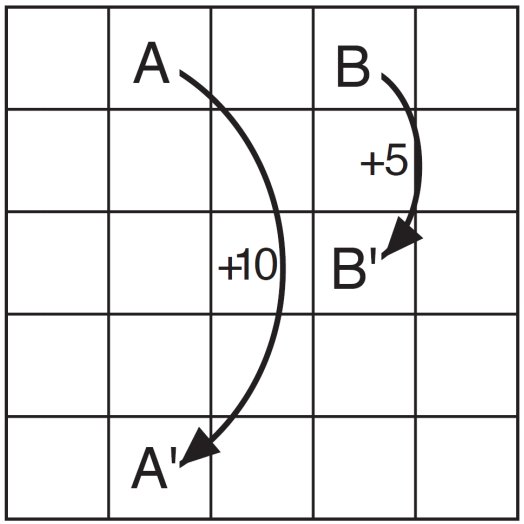
$$
V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')\right)
$$

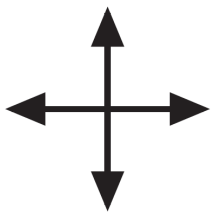
This is a version of Bellmann's equation. A fundamental property of value functions is that they satisfy a set of recursive consistency equations. Crucially V^{π} has unique solution.

Policy Evaluation

1. Let us consider how to compute the value function for an arbitrary policy. This is called policy evaluation in the DP literature. We also refer to it as the prediction problem.
2. We could simply systematically apply Bellman's equation over and over again to obtain better estimates for $V_1(s), V_2(s), \ldots, V_k(s)$ iteratively, hoping that the solution converges (it is guaranteed to do so, more on this later). This algorithm is called iterative policy evaluation. Each iteration step k provides successively better approximations to the solution.
3. A typical stopping condition for iterative policy evaluation is to measure by how little the largest change in value function was between two iteration steps and set a cut-off threshold.

Gridworld example





Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Values for random actions

Iterative Policy Evaluation Algorithm

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$

Iterative Policy Evaluation Algorithm

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$

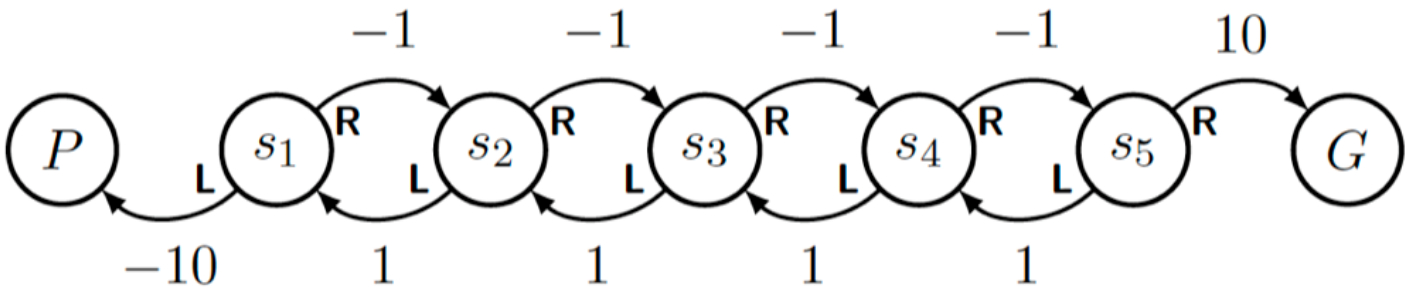
In Iterative Policy evaluation we sweep through all successor states, we call this kind of operation a full backup.

Iterative Policy Evaluation Algorithm

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$

In Iterative Policy evaluation we sweep through all successor states, we call this kind of operation a full backup.
To produce each successive approximation V_{k+1} from V_k iterative policy evaluation applies the same operation to each state s: it replaces the old value of s in place with a new value obtained from the old values of the successor states of s, and the expected immediate rewards, along **all** the one-step transitions possible under the policy being evaluated. We could also run a code version storing old and new arrays for V. This turns out to converge slower, why?

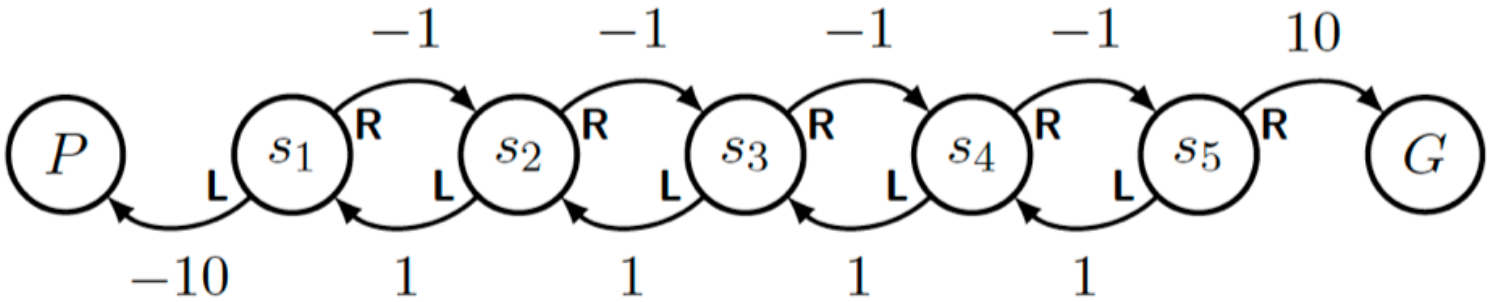
Stair Climbing MDP



Example (Stair Climbing)

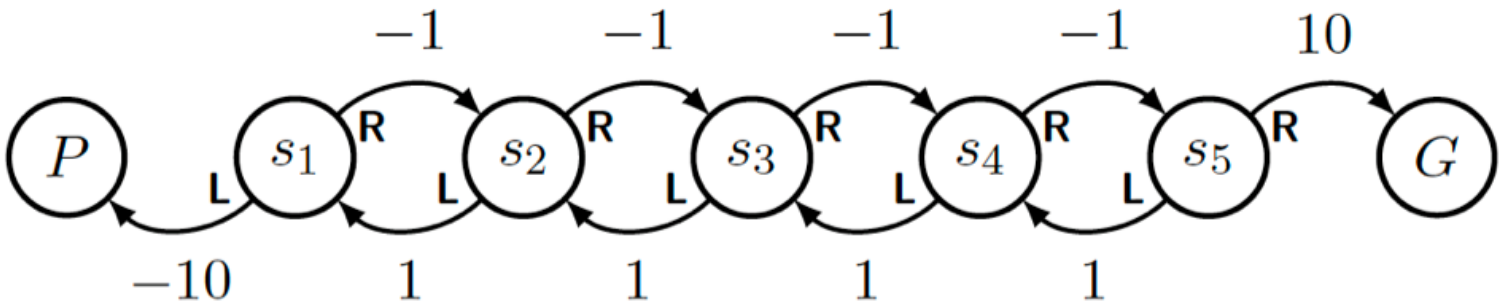
Actions are *Left* and *Right* and states are $P, s_1, \ldots, s_5, G$. States P and G are absorbing and we start in s_3. $\gamma = 0.9$. Let us assume an unbiased policy, where every action is equally probable $\pi(s, L) = \frac{1}{2} = \pi(s, R)$. We want now to evaluate the value function $V^\pi(s)$ for each state in s.

Policy evaluation: Stair climbing 1



$\hat{V}_0$:	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$\hat{V}_1$:							

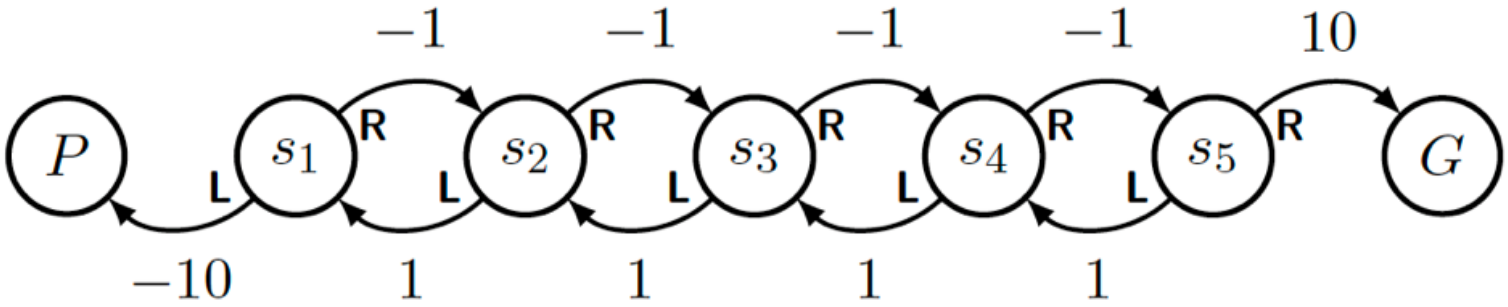
Policy evaluation: Stair climbing 2



$\hat{V}_0$: 0.0 0.0 0.0 0.0 0.0 0.0 0.0

$\hat{V}_1$: 0.0 −5.5

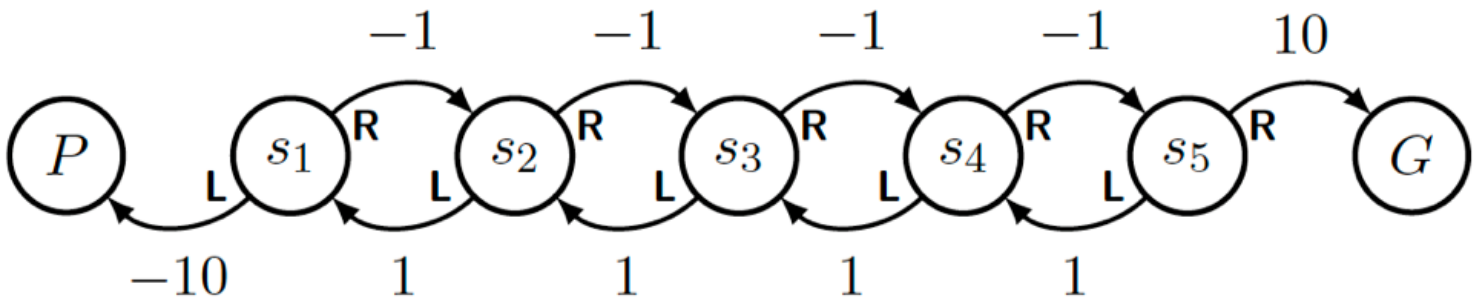
Policy evaluation: Stair climbing 3



$\hat{V}_0$: 0.0 0.0 0.0 0.0 0.0 0.0 0.0

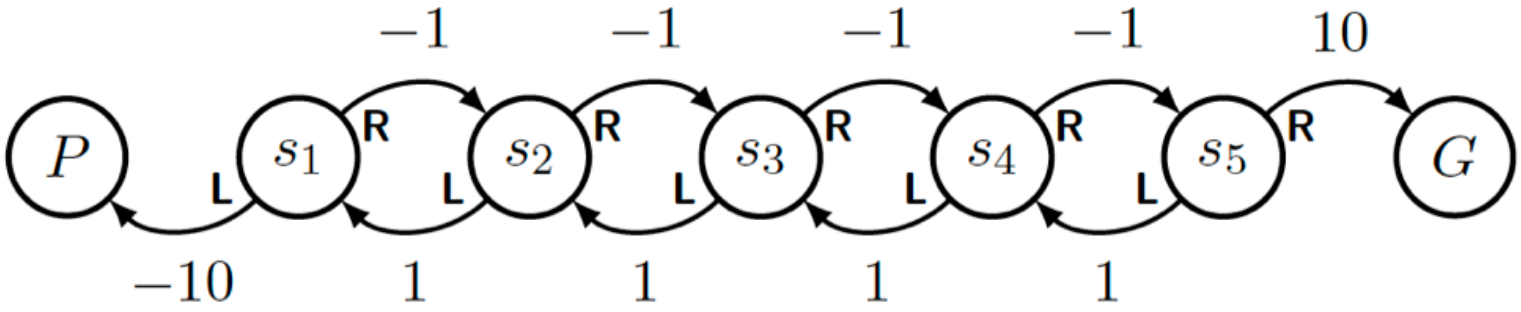
$\hat{V}_1$: 0.0 −5.5 0.0

Policy evaluation: Stair climbing 4



$\hat{V}_0$:	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$\hat{V}_1$:	0.0	−5.5	0.0	0.0	0.0	5.5	0.0

Policy evaluation: Stair climbing 5



$\hat{V}_0$:	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$\hat{V}_1$:	0.0	−5.5	0.0	0.0	0.0	5.5	0.0
$\hat{V}_2$:	0.0	−5.5	−2.48	0.0	2.48	5.5	0.0
$\hat{V}_3$:	0.0	−6.61	−2.48	0.0	2.48	6.61	0.0
$\hat{V}_4$:	0.0	−6.61	−2.98	0.0	2.98	6.61	0.0
$\hat{V}_\infty$:	0.0	−6.90	−3.10	0.0	3.10	6.90	0.0

State-Action Value function as Cost-To-Go

How good is it to be in a given state and take a given action when you follow a policy π:

Definition (State-Action Value function "Cost to Go")

$$Q^{\pi}(s,a) = \mathbb{E}\left[R_t | S_t = s, A_t = a\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s, A_t = a\right] \tag{20}$$

State-Action Value function as Cost-To-Go

How good is it to be in a given state and take a given action when you follow a policy π:

Definition (State-Action Value function "Cost to Go")

$$Q^{\pi}(s,a) = \mathbb{E}\left[R_t | S_t = s, A_t = a\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s, A_t = a\right] \tag{20}$$

The relation between (state) value function and the state-action value function is straightforward:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(s,a) Q^{\pi}(s,a) \tag{21}$$

Optimal Value and Cost-to-Go function for MDPs

Value functions define a partial ordering over policies. A policy is defined to be better than or equal to a policy if its expected return is greater than or equal to that of for all states. In other words, $\pi \geq \pi'$ if and only if $V^{\pi}(s) \geq V^{\pi'}(s)$ for all $s \in \mathcal{S}$.

Definition (Optimal Value function)

$$V^{*}(s) = \max_{\pi} V^{\pi}(s), \forall s \in \mathcal{S} \tag{22}$$

Optimal Value and Cost-to-Go function for MDPs

Value functions define a partial ordering over policies. A policy is defined to be better than or equal to a policy if its expected return is greater than or equal to that of for all states. In other words, $\pi \geq \pi'$ if and only if $V^{\pi}(s) \geq V^{\pi'}(s)$ for all $s \in \mathcal{S}$.

Definition (Optimal Value function)

$$V^{*}(s) = \max_{\pi} V^{\pi}(s), \forall s \in \mathcal{S} \tag{22}$$

Therefore, the policy π^{*} that maximises the value function is the optimal policy. There is always at least one optimal policy. There may be more than one optimal policy.

Optimal Value and Cost-to-Go function for MDPs

Value functions define a partial ordering over policies. A policy is defined to be better than or equal to a policy if its expected return is greater than or equal to that of for all states. In other words, $\pi \geq \pi'$ if and only if $V^{\pi}(s) \geq V^{\pi'}(s)$ for all $s \in \mathcal{S}$.

Definition (Optimal Value function)

$$V^*(s) = \max_{\pi} V^{\pi}(s), \forall s \in \mathcal{S} \tag{22}$$

Therefore, the policy π^* that maximises the value function is the optimal policy. There is always at least one optimal policy. There may be more than one optimal policy.

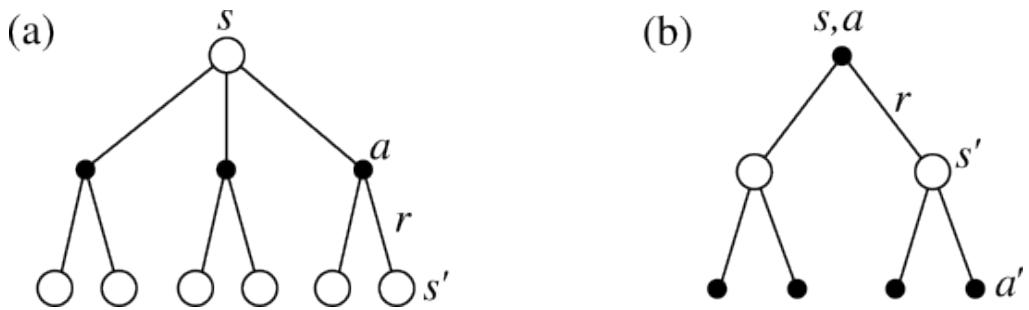
Definition (Optimal State-Action Value function)

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{23}$$

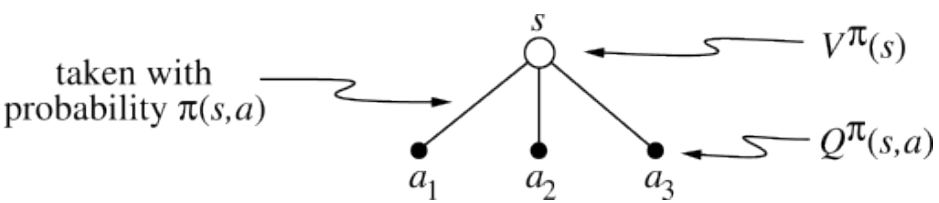
In analogy to before we also have

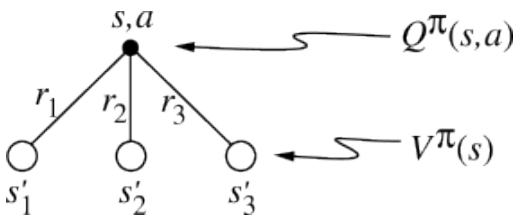
$$Q^*(s, a) = \mathbb{E}\left[r_{t+1} + \gamma V^*(s_{t+1}) | S_t = s, A_t = a\right] \tag{24}$$

Bellman Equation Backups



Value function $V^{\pi}(s)$ (Fig a) and State-Action Value function $Q^{\pi}(s, a)$ (Fig b) backup diagrams. Relationship between these and the V and Q functions in action:





Bellman optimality equation (derivation)

The value function V^{π^*} for a policy π^* must satisfy the self-consistency condition given by the Bellman equation. Moreover, if we want to have the optimal value for a state, we should only choose the action(s) that yield the highest value, $V(s) = \sum_{a \in \mathcal{A}} \pi(s,a) Q^{\pi}(s,a)$ should become $\max_{a \in \mathcal{A}} Q^{\pi}(s,a)$:

$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}} \sum_{a \in \mathcal{A}} \pi(s,a) Q^{\pi}(s,a) \\
&= \max_{a \in \mathcal{A}} Q^{\pi^*}(s,a) \\
&= \max_{a} \mathbb{E}\left[R_t | S_t = s, A_t = a\right] \\
&= \max_{a} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s, A_t = a\right] \\
&= \max_{a} \mathbb{E}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | S_t = s, A_t = a\right] \\
&= \max_{a} \mathbb{E}\left[r_{t+1} + \gamma V^*(s_{t+1}) | S_t = s, A_t = a\right] \\
&= \max_{a} \sum_{s'} P\left[s' | s, a\right] \left(r(s,a,s') + \gamma V^*(s')\right) \\
&= \max_{a} \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V^*(s')\right)
\end{aligned}
$$

Bellman optimality equation for V^*

V^{π^*} is the optimal value function and so conveniently the self-consistency condition can be rewritten in a form without reference to any specific policy π^*: $V^{\pi^*} = V^*$. This yields the Bellman Optimality Equation for an optimal policy:

Definition (Bellmann Optimality Equation for V)

$$V^*(s) = \max_a \sum_{s'} P\left[s'|s,a\right]\left(r(s,a,s') + \gamma V^*(s')\right) \quad (25)$$

$$= \max_a \sum_{s'} \mathcal{P}^a_{ss'}\left(\mathcal{R}^a_{ss'} + \gamma V^*(s')\right) \quad (26)$$

Intuitively, the Bellman Optimality Equation expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state.

Bellman optimality equation for Q^*

Definition (Bellman Optimality Equation for Q^*)

$$Q^*(s,a) = \mathbb{E}\left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \middle| S_t{=}s, A_t{=}a\right] \tag{27}$$

$$= \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a')\right) \tag{28}$$

Bellman optimality equation for Q^*

Definition (Bellman Optimality Equation for Q^*)

$$Q^*(s,a) = \mathbb{E}\left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \middle| S_t{=}s, A_t{=}a\right] \quad (27)$$

$$= \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a')\right) \quad (28)$$

Note how the Bellmann Optimality Equation does not need $\pi^*(a,s)$ explicitly to compute $V^*(s)$ – this is very useful (Why?).

Bellman optimality equation for Q^*

Definition (Bellman Optimality Equation for Q^*)

$$Q^*(s,a) = \mathbb{E}\left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | S_t{=}s, A_t{=}a\right] \quad (27)$$

$$= \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a')\right) \quad (28)$$

Note how the Bellmann Optimality Equation does not need $\pi^*(a,s)$ explicitly to compute $V^*(s)$ – this is very useful (Why?).
For finite MDPs, the Bellman optimality equation has a unique solution independent of the policy. Why?

Bellman optimality equation for Q^*

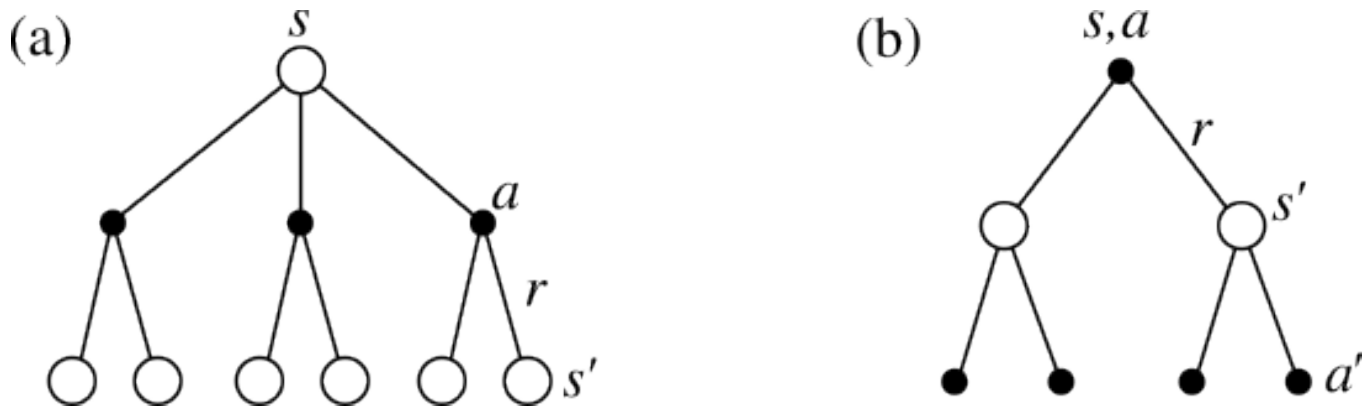
Definition (Bellman Optimality Equation for Q^*)

$$Q^*(s,a) = \mathbb{E}\left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | S_t{=}s, A_t{=}a\right] \quad (27)$$

$$= \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a')\right) \quad (28)$$

Note how the Bellmann Optimality Equation does not need $\pi^*(a,s)$ explicitly to compute $V^*(s)$ – this is very useful (Why?).
For finite MDPs, the Bellman optimality equation has a unique solution independent of the policy. Why?
The Bellman optimality equation is a set of N non-linear equations, where $N = |\mathcal{S}|$, with N unknowns. Because for known $\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}$ one can solve this system of equations for using any method for solving systems of nonlinear equations.

Bellman Optimality Equation Backups



Value function $V(s)$ (a) and State-Action Value function $Q(s, a)$ (b) backup diagrams.

On solving the Bellman Optimality Equations

Explicitly solving the Bellman optimality equation provides one route to finding an optimal policy, and thus to solving the reinforcement learning problem. This solution approach can often be challenging at best, if not impossible, because it is like an exhaustive search, looking ahead at all possible traces, computing their probabilities of occurrence and their desirabilities in terms of expected rewards. Moreover, this solution relies on at least 3 assumptions that are rarely true in practice:

- we accurately know the dynamics of the environment
- we have computational resources to find the solution
- the Markov property.

Thus, in reinforcement learning often we have to (and want to) settle for approximate solutions.

Bellman Optimality Equation convergence

Theorem (Bellman Optimality Equation convergence theorem)

For an MDP with a finite state and action space

1. *The Bellman (Optimality) equations have a unique solution.*
2. *The values produced by value iteration converge to the solution of the Bellman equations.*

A complete proof of this theorem is beyond the scope of this course. It rests on a very useful lemma from analysis called the Banach Fixed Point Theorem, a.k.a the Contraction Mapping Theorem. But we can gain some intuition of the proof.

Convergence proof I

Intuition for the proof

I. the Bellman Optimality Equation (BOE) $V^{\pi^*}(s) = \max_{a \in \mathcal{A}} \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V^{\pi^*}(s') \right)$ is a so called fixed point equation.

II. Suppose we start with some arbitrary value function $V(\dots)$. We can apply the right hand side of the BOE to $V(\dots)$ to get a new value function $V'(\dots)$. Thus, the right hand side of the BOE defines an operator K on the space of value functions (taking a function as argument and yielding a function).

III. The Bellman optimality equation states that the optimal value function is a fixed point: $V^{\pi^*} = K(V^{\pi^*})$.

IV. A simple process to find a fixpoint of K is as follows:

1. Begin with any value function V_0, e.g. $V_o(s) = 0 \forall s$.
2. Apply K to it to get V_1: $V_1 \leftarrow K(V_0)$

Convergence proof II

3. Apply K again to get V_2: $V_2 \leftarrow K(V_1)$
4. ...

This is in fact what value iteration does.

V. This process of iterating K converges to a limit and the limit is unique. Why? K has a special property: it is a contraction. This means that given any two value functions V_1 and V_2, applying to them K brings them closer together:

$$\|K(V_1) - K(V_2)\|_\infty \leq \gamma \|V_1 - V_2\|_\infty \tag{29}$$

where $V_1 \neq V_2$ and norm $\|\ldots\|$ is the sup norm/max norm

$$\|V\|_\infty = \max_{s \in \mathcal{S}} |V(s)| \tag{30}$$

Convergence proof III

VI The constant $\gamma \in [0, 1[$ is the MDP's discount factor. With $\gamma < 1$, the norm between two distinct points decreases strictly. Intuitive reasoning: if we have a process that keeps bringing points closer to each other, and we iterate the process, then we will converge to a fixed point.

VII The Contraction Mapping Theorem says that under appropriate conditions, which are satisfied by the BOE and MDPs, this is in fact the case: the process of iterating K has a limit, which is the unique fixed point of K: V^{π^*}.

Exercises

- Exercise 3.1
- Exercise 3.2
- Exercise 3.4
- Exercise 3.7
- Exercise 3.8
- Exercise 3.11
- Exercise 3.12
- Exercise 3.13
- Exercise 3.14
- Exercise 3.17
- Exercise 3.19
- Exercise 3.22
- Exercise 3.25
- Exercise 3.26
- Exercise 3.27
- Exercise 3.28